second preferred embodiment of the present invention provides a more generalized mechanism for non-default drag and drop operation.

The second preferred embodiment of the present invention enables the user to perform a conventional drag and drop operation or, alternatively, a non-default drag (enhanced) and drop operation. The second preferred embodiment is applicable to any drag and drop operation and is not limited to drag and drop operations that involve text.

The second preferred embodiment of the present invention is especially well adapted for use in an object oriented programming environment. Hence, in the second preferred embodiment of the present invention described herein, the code for the operating system **50** provides an object-oriented programming environment. The target and source of the drag and drop operation are implemented as "objects." An object is a combination of data members (i.e., attributes held in data structures) and member functions that act upon the data members. Those skilled in the art will appreciate, however, that the second preferred embodiment of the present invention need not be implemented in an object-oriented programming environment.

FIG. **12** is a flowchart illustrating the steps that are performed in a conventional drag and drop operation using the primary button **12** of the mouse **10** (e.g., the left button in this case). Initially, a user moves the mouse **10** (FIG. **1**) to move the cursor **16** (FIG. **2**) on the video display **18** until the cursor is positioned over a source-visual element **20** that is associated with the source object. The user then clicks on the source-visual element **20** by depressing button **12** of the mouse **10** (step **126** in FIG. **12**). While keeping the button **12** depressed, the user drags the source-visual element **20** (FIG. **2**) across the video display **18** until the source-visual element **20** is positioned over a target visual element **22** that is associated with the target object (step **128** in FIG. **12**). The user then releases the left button **12** of the mouse **10** to cause a drop in the source-visual element **20** onto the target-visual element **22** (step **130** in FIG. **12**). A default operation is automatically performed (step **132**), and the source-visual element **20** is advised of the default operation (step **134**).

The non-default or expanded drag and drop operation of the second preferred embodiment of the present invention is performed by executing the steps shown in the flowchart of FIG. **13**. The expanded drag and drop operation is initiated by positioning the cursor **16** (FIG. **2**) on the source-visual element **20** and clicking the secondary button **14** (FIG. **1**) of the mouse **10** (step **152** in FIG. **13**). The source-visual element **20** in the target-visual element may take many forms, including text, icons or graphic elements. The code for the source object **46** (FIG. **3**) is informed that the cursor **16** has been positioned over the source-visual element and is also informed that the secondary button **14** (FIG. **1**) of the mouse **10** has been depressed by the messages that are sent to it. Movement of the mouse **10** and depressing a mouse button are events that are translated into messages by the code for the operating system **50**. These messages are placed in a message queue for the executing application program.

Each application program run on the operating system has a separate message queue. Each application program retrieves messages from the message queue using a selection of code known as the "message loop." Once the application program has retrieved a message from its message queue, the application program determines which window procedure should receive the message. A separate window procedure is provided for each window. The application program may provide several windows, and thus, the application program must determine which window is the proper destination for the message when the message is retrieved from the queue.

When the cursor **16** is positioned over the source-visual element **20**, a message is generated that specifies the position of the cursor and the window which holds the source visual element **20**. This message provides a relative position of the cursor **16** in (X,Y) coordinates relative to the upper left-hand corner of the window. The window procedure associated with the window that holds the source-visual element **20** receives this message and passes the message to the code for the source object **46** (FIG. **3**). When the secondary button **14** (FIG. **1**) of the mouse **10** is depressed, the code for the source object **46** receives a message informing of the depression of the secondary button.

Once step **152** of FIG. **13** is performed, the source visual element **20** (FIG. **2**) is dragged by movement of the mouse **10** (FIG. **1**) until the source-visual element is positioned over the target visual element **22**. As the drag begins, the code for the source object **46** sends a message to the code for the operating system **50**. This message holds an identifier for the source object associated with the source-visual element, information about the source-visual element **20** and an indication of the operations that may be performed on the source object (step **154** in FIG. **13**). The code the operating system **50** is responsible for tracking the mouse movements until the source-visual element **20** is dropped.

When the source-visual element **20** is positioned over the target-visual element **22**, the secondary button **14** of the mouse **10** is released to drop the source visual element (step **156** in FIG. **13**). The code for the operating system **50** (FIG. **3**) for the list of valid operations from the source object for code to the target object **48**. A context menu **163**, such as shown in FIG. **14**, is then displayed to show the possible operations that a user may perform (step **158** in FIG. **13**). The operations listed on the context menu may include operations such as "move," "copy," and "link." Among the operations shown in the context menu **163** is the default operation, which is indicated in boldface (note that "move" is boldfaced in FIG. **14**). The operations listed in the context menu **163** depend upon the nature of the target object and the source object. The determination of what operations are listed on the context menu **163** will be described in more detail below. The user then may choose an operation for the context menu **163** (step **160** in FIG. **13**). The system determines that the selection is chosen and determines the identity of any such user-selected operation (step **161**). The selection option is then performed (step **162**). The user may also cancel the effect of the drag and drop operation so that no operation is performed. For example, a canceled option may be provided on the context menu **162** (FIG. **14**) or the user may cancel the operation by hitting the escape button or clicking the mouse outside the menu.

In order to understand what occurs once the source-visual element **20** (FIG. **2**) is positioned over the target-visual element **22** in the second preferred embodiment of the present invention, it is helpful to review the steps performed by the code for the target object **48** (FIG. **3**).

FIG. **15** is a flowchart showing the steps performed by the target object for a typical drag and drop sequence in the second preferred embodiment of the present invention. Before a target object may be a target for a drag and drop operation, it must first register as a target (step **154**). If an object is not registered as a target object, a source object will not be allowed to drop on the target object. Later, once the